

# CSE 464: Software Quality Assurance and Testing

## Course Project Part-2

Vajravel Conjeevaram Manigandan

1225428152

[vconjeev@asu.edu](mailto:vconjeev@asu.edu)

### Introduction:

This is a Java based application that can parse graphs in DOT format, manipulate graphs (e.g., add nodes, add edges), and output graphs in DOT format and graphics (e.g., jpg and png).

This project was built using **IntelliJ** as the IDE, **Java 21** as the Java Development Kit (JDK), **Maven** to manage dependencies, and the **graphviz-java** was used to parse the DOT file into objects in java and vice versa. Classes such as Graph Manager, Graph, Node, and Edge were implemented to perform operations such as coding the features, creating a graph, adding nodes or edges on the given graph. Also the Path class was added to implement new features such as Breadth-first-search(BFS) and Depth-first-search(DFS).

**NOTE: *If there are any dependency issues, please resolve them using mvn install in the command line terminal or resolve them using maven in IntelliJ.***

### Access the project:

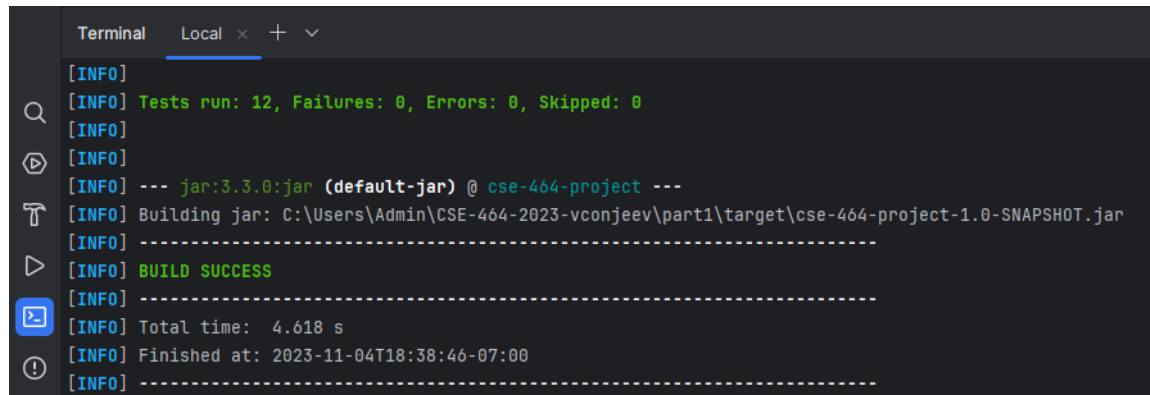
Every step of the project has been tracked using Github, and can be accessed via the link:  
<https://github.com/v-Cm/CSE-464-2023-vconjeev>

### Run the project:

Clone or download the project folder to your system. If you are using an IDE (IntelliJ) to access the project, you can simply run the **Application.java** file in the **/part1/src/main/java/** path. Or, if you are using the CLI/terminal you can type **mvn package** to compile and run the project.

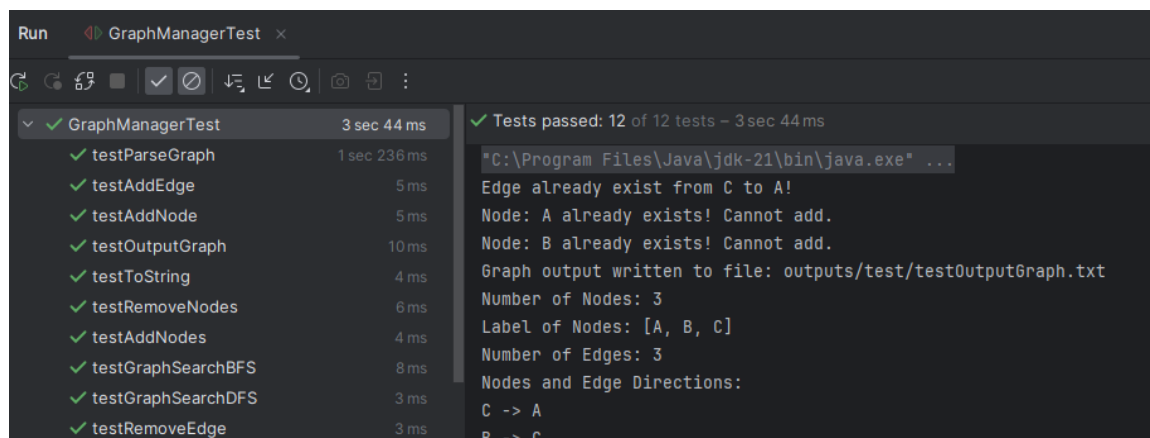
### Test the project:

The application has several features in it (which will be described in detail in the following section) and the expected outputs for each features have been provided either in the **GraphManagerTest.java** file in the **/part1/test/** path, or as files in the **/part1/outputs/expected** directory. Again, you can run **mvn test** or **mvn package** on the terminal,



```
[INFO]
[INFO] Tests run: 12, Failures: 0, Errors: 0, Skipped: 0
[INFO]
[INFO]
[INFO] --- jar:3.3.0:jar (default-jar) @ cse-464-project ---
[INFO] Building jar: C:\Users\Admin\CSE-464-2023-vconjeev\part1\target\cse-464-project-1.0-SNAPSHOT.jar
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 4.618 s
[INFO] Finished at: 2023-11-04T18:38:46-07:00
[INFO] -----
```

or run the testcases in the **GraphManagerTest.java** file manually.



```
Run GraphManagerTest x
✓ GraphManagerTest 3 sec 44 ms
  ✓ testParseGraph 1 sec 236 ms
  ✓ testAddEdge 5 ms
  ✓ testAddNode 5 ms
  ✓ testOutputGraph 10 ms
  ✓ testToString 4 ms
  ✓ testRemoveNodes 6 ms
  ✓ testAddNodes 4 ms
  ✓ testGraphSearchBFS 8 ms
  ✓ testGraphSearchDFS 3 ms
  ✓ testRemoveEdge 3 ms
  ✓ Tests passed: 12 of 12 tests - 3 sec 44 ms
  *C:\Program Files\Java\jdk-21\bin\java.exe* ...
  Edge already exist from C to A!
  Node: A already exists! Cannot add.
  Node: B already exists! Cannot add.
  Graph output written to file: outputs/test/testOutputGraph.txt
  Number of Nodes: 3
  Label of Nodes: [A, B, C]
  Number of Edges: 3
  Nodes and Edge Directions:
  C -> A
  B -> C
```

Five new testcases have been added to check all the features added in this part of the project.

### About the features:

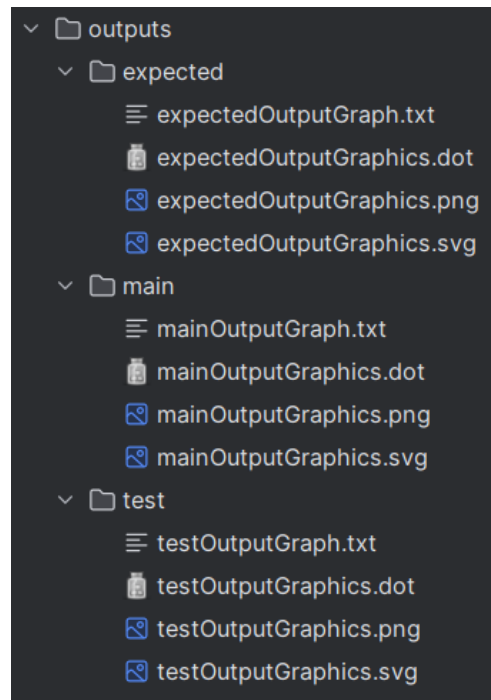
All the features required in part 2 are added into a Class named “GraphManager.java” and implemented in “Graph.java”, “Path.java”, and “Algorithm.java” which are in the folder: part1/src/main/java.

**Input:** The input DOT file is **testInput.dot** (which can be edited however you want).

```
1 digraph {
2     A -> B;
3     B -> C;
4     C -> A;
5 }
```

testInput.dot

**Output:** All the outputs are generated in the outputs directory as detailed below:



Outputs directory

- expected - contains the expected output files
- main - contains the outputs generated by running the main function in Application.java
- test - contains the outputs generated by the testcases in GraphManager.java

**Feature 1:** Support removing nodes and edges.

[Github commit link.](#)

This feature implements three APIs:

- Remove a node API: `removeNode(String label)`
- Remove nodes API: `removeNodes(String[] label)`
- Remove an edge API: `removeEdge(String srcLabel, String dstLabel)`

```
graphManager.removeNode( label: "F");  
graphManager.toString();  
  
graphManager.removeNodes(new String[]{"E", "D"});  
graphManager.toString();  
  
graphManager.removeEdge( srcLabel: "D", dstLabel: "G");  
graphManager.toString();
```

Implementation of the APIs

```

Number of Nodes: 7
Label of Nodes: [A, B, C, D, E, G, I]
Number of Edges: 4
Nodes and Edge Directions:
C -> A
D -> G
B -> C
A -> B
Number of Nodes: 5
Label of Nodes: [A, B, C, G, I]
Number of Edges: 3
Nodes and Edge Directions:
C -> A
B -> C
A -> B
No such edge exist from D to G
Number of Nodes: 5
Label of Nodes: [A, B, C, G, I]
Number of Edges: 3
Nodes and Edge Directions:
C -> A
B -> C
A -> B

```

Output of the above APIs as executed in Application.java

## Feature 2: Adding continuous integration.

[Github commit link.](#)

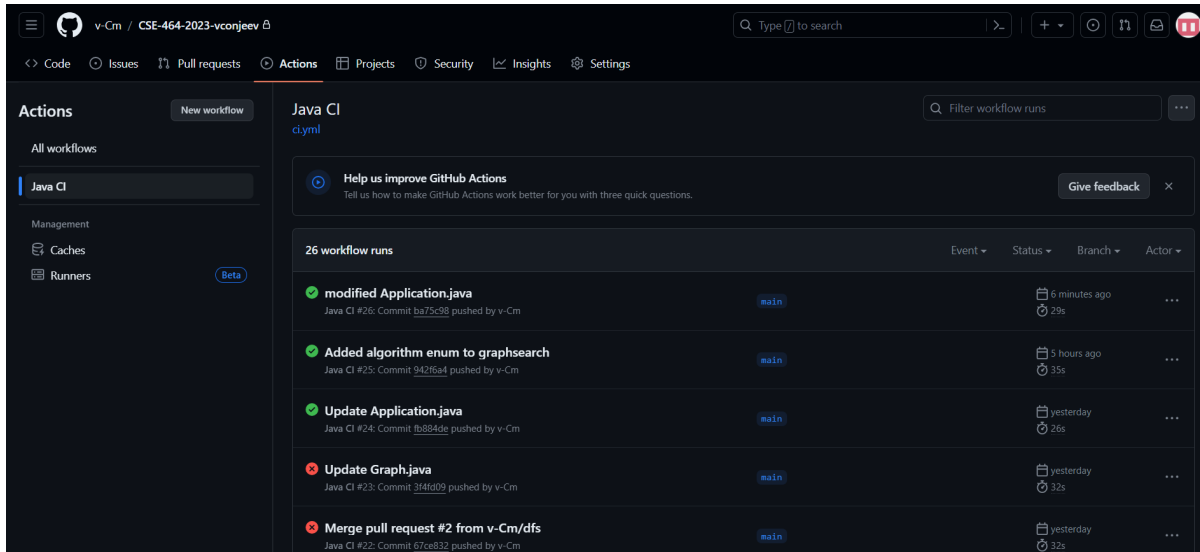
[Github Actions Workflow link](#) (ci.yml)

The ci.yml file builds and tests the code every time there is a new push.

The screenshot shows the GitHub Actions interface for a workflow named 'build'. The workflow is in a 'succeeded' state, having completed 1 minute ago. The left sidebar shows the 'Summary' tab selected, with a list of jobs including 'build'. The main area displays the 'build' job details, showing a list of steps and their durations:

Step	Duration
Set up job	1s
Run actions/checkout@v3	1s
Set up Java 21	7s
Build and test with Maven	11s
Post-Set up Java 21	8s
Post-Run actions/checkout@v3	8s
Complete job	8s

Output after running CI after a push



The implemented workflow on Github actions

### Feature 3: Create a branch BFS and implement BFS.

[Github commit link.](#)

[BFS branch commits link](#)

This feature implements one API:

- Path GraphSearch(Node src, Node dst) API: accepts the source and destination node to find the path using BFS. If no path exists, it will return null.
- The class Path has been created in "Path.java" in the BFS branch.

```
path = graphManager.GraphSearch(new Node("A"), new Node("A"));
System.out.println(path != null ? "The path:" + path : "No path exists.");

path = graphManager.GraphSearch(new Node("B"), new Node("A"));
System.out.println(path != null ? "The path:" + path : "No path exists.");
```

Invoking the GraphSearch API

```
The path:A
The path:B -> C -> A
```

The available paths displayed after running the BFS API

### Feature 4: Create a branch DFS and implement DFS.

[Github commit link.](#)

[DFS branch commits link](#)

This feature implements one API:

- Path GraphSearch(Node src, Node dst) API: accepts the source and destination node to find the path using DFS. If no path exists, it will return null.
- The class Path has been created in "Path.java" in the DFS branch.

```
path = graphManager.GraphSearch(new Node("A"), new Node("D"));
System.out.println(path != null ? "The path:" + path : "No path exists.");

path = graphManager.GraphSearch(new Node("B"), new Node("A"));
System.out.println(path != null ? "The path:" + path : "No path exists.");
```

Invoking the GraphSearch API

```
No path exists.
The path:B -> C -> A
```

The available paths displayed after running the DFS API

**Feature 5:** Merge the BFS and DFS branches to the main branch.

[Github commit link.](#)

[Main branch commits link](#)

This feature is to resolve the merge conflict caused by merging the three branches by changing the GraphSearch API to add an enum parameter for choosing search algorithm.

- Path GraphSearch(Node src, Node dst, Algorithm algo): accepts either BFS or DFS as parameter and calls the corresponding API.
- The Algorithm enum has been implemented in "Algorithm.java".

```
path = graphManager.GraphSearch(new Node("B"), new Node("A"), Algorithm.BFS);
System.out.println(path != null ? "The path:" + path : "No path exists.");

path = graphManager.GraphSearch(new Node("A"), new Node("A"), Algorithm.DFS);
System.out.println(path != null ? "The path:" + path : "No path exists.");
```

Invoking the GraphSearch API along with the algorithm

```
The path:B -> C -> A
The path:A
```

Output of the API calls above.

**Pull requests:** There were two pull requests created and closed to merge the BFS branch to the Main followed by the DFS branch.

- [Merge BFS to main link](#)

### Merge bfs #1

**Merged** v-Cm merged 3 commits into `main` from `bfs` yesterday

Conversation 0 Commits 3 Checks 1 Files changed 26

**v-Cm** commented yesterday

merge bfs branch to main branch.

**v-Cm** added 3 commits 2 days ago

- created bfs branch and implemented bfs search ✗ d970b2d
- adding Path.java ✓ d5fd91c
- created test cases ✓ a0889ec

**v-Cm** merged commit `a88b178` into `main` yesterday  
1 check passed

**Pull request successfully merged and closed**  
You're all set—the `bfs` branch can be safely deleted.

[Delete branch](#)

[View details](#) [Revert](#)

- [Merge DFS to main link](#)

### Dfs #2

**Merged** v-Cm merged 4 commits into `main` from `dfs` yesterday

Conversation 0 Commits 4 Checks 1 Files changed 10

**v-Cm** commented yesterday

Merging dfs to main

**v-Cm** and others added 4 commits 2 days ago

- some error ✓ abff366
- some errorng ✓ b668da7
- implemented dfs function ✓ a3a2063
- Merge branch "main" into dfs Verified ✗ 1861c65

**v-Cm** merged commit `67ce832` into `main` yesterday  
1 check failed

**Pull request successfully merged and closed**  
You're all set—the `dfs` branch can be safely deleted.

[Delete branch](#)

[View details](#) [Revert](#)